# Practical-1 (Introduction to Linux Commands)

---> Is practical me koi code nahi hai bhai

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-2 (Shell Programming)

Q1. Write a shell script to write "Multiplication Tables"

---> Code:

#! /usr/bin/bash

echo "Enter a Number: "

read a

for((i=0; i<a; i++))

do

b=$((a\*i))

echo "$a x $i = $b"

done

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q2. Write a shell script for a "Small Calculator"

---> Code:

#! /usr/bin/bash

echo "Enter 1st Number: "

read a

echo "Enter 2nd Number: "

read b

echo "Which operation do you wanna perform"

echo "1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit Calculator Interface"

read c

if [ $c -eq 1 ]

then

let d=$a+$b

echo "Sum of these two numbers is: $d"

fi

if [ $c -eq 2 ]

then

echo " "

echo "Do you wanna Subtract: "

echo "1. 1st no. from 2nd no."

echo "OR"

echo "2. 2nd no. from 1st no."

echo "Enter 1 or 2 as per your need."

read e

if [ $e -eq 1 ]

then

let d=$b-$a

echo " "

echo "When you subtract 1st no. from 2nd no. you get: $d"

elif [ $e -eq 2 ]

then

let d=$a-$b

echo " "

echo "When you subtract 2nd no. from 1st no. you get: $d"

fi

fi

if [ $c -eq 3 ]

then

let d=$a\*$b

echo "Multiplication of these two numbers is: $d"

fi

if [ $c -eq 4 ]

then

echo " "

echo "Do you wanna Divide: "

echo "1. 1st no. by 2nd no."

echo "OR"

echo "2. 2nd no. by 1st no."

echo "Enter 1 or 2 number as per your need."

read e

if [ $e -eq 1 ]

then

echo " "

echo "When you divide 1st no. by 2nd no. you get: "

echo "scale=2; $a/$b" | bc

elif [ $e -eq 2 ]

then

echo " "

echo "When you divide 2nd no. by 1st no. you get: "

echo "scale=2; $b/$a" | bc

fi

fi

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q3. Write a shell script for "Displaying prime numbers upto a given limit"

---> Code:

#! /usr/bin/bash

echo "Enter a Number: "

read a

for ((i=2; i<=a; i++))

do

n=0

for ((j=2; j<i; j++))

do

let b=$i%$j

if [ $b -eq 0 ]

then

n=1

fi

done

if [ $n -eq 0 ]

then

echo $i

fi

done

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-3 (File Manipulation Using System Calls)

Q1. Using system calls copy first half of the content of a already existing file to a newly created file and then again copy the rest remaining second half of the content of that older file to a another newly created file

---> Code:

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main(){

int a, b, c, n;

char buff1[500];

a = open("content.txt", O\_RDONLY | O\_CREAT, 0777);

b = open("FirstHalf.txt", O\_WRONLY | O\_CREAT, 0777);

c = open("SecondHalf.txt", O\_WRONLY | O\_CREAT, 0777);

n = read(a, buff1, 500);

read(a, buff1, n/2);

write(b, buff1, n/2);

lseek(a, n/2, SEEK\_SET);

read(a, buff1, n/2);

write(c, buff1, n/2);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q2. Using system calls write a program which reads from console until user types '$' and the content which is written on the console before '$' copy that content to a newly created file

---> Code:

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main(){

int a, b=0;

char buff1[500];

scanf("%[^\n]s", buff1);

a = open("Output$.txt", O\_WRONLY | O\_CREAT, 0777);

for(int i=0; i<500; i++){

if(buff1[i] == '$'){

printf("You can not write after '$' Symbol \n");

break;

}

else{

b++;

}

}

char buff2[b];

for(int i=0; i<b; i++){

buff2[i]=buff1[i];

}

write(a, buff2, b);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q3. Write a program using system call to read the contents of a file without using char array and display the contents on the console (Don't use any built in functions like sizeof() and strlen())

---> Code:

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main(){

int a, b;

a = open("Input.txt", O\_RDONLY | O\_CREAT, 0777);

char buff[1];

char \*c = buff;

while((b = read(a, c, 1))>0){

write(1, c, 1);

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-4 (Directory Manipulation Using System Calls)

Q1. Write a program using directory system calls, make a directory on desktop and create a file inside the directory and list the contents of the directory

---> Code:

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<dirent.h>

int main()

{

int a = mkdir("My\_Directory",0777);

if(a!=-1)

{

printf("# You're directory has been created: \n");

printf(" \n");

}

struct dirent \*dptr;

int fd1 = creat("My\_Directory/testing\_123.txt",0777);

int fd2 = creat("My\_Directory/noicee.txt",0777);

int fd3 = creat("My\_Directory/life\_is\_good.txt",0777);

if(fd1!=-1 && fd2!=-1 && fd3!=-1)

{

printf("\* file testing\_123.txt is created.\n");

printf("\* noicee.txt is created.\n");

printf("\* life\_is\_good.txt is created.\n");

printf(" \n");

}

DIR \*dp = opendir("My\_Directory");

printf("@ List of files in created directory: \n");

while(NULL!=(dptr = readdir(dp)))

{

printf("%s\n", dptr->d\_name);

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q2. Write a program using directory and file manipulation system calls to copy the contents of one directory to a newly created directory

---> Code:

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<dirent.h>

int main()

{

struct dirent \*dptr;

int a = mkdir("My\_Directory", 0777);

int b = mkdir("Your\_Directory", 0777);

if(a==0 && b==0){

printf("\* Directories named 'My\_Directory' and 'Your\_Directory' has been created Successfully. \n");

}

else{

printf("\* Either directories named 'My\_Directory' & 'Your\_Directory' already exist or they were not able to create due to any Error. \n");

}

int fd1 = open("My\_Directory/My\_File.txt", O\_CREAT|O\_RDWR, 0777);

printf("\* File named 'My\_File.txt' has been created Successfully in Directory named 'My\_Directory'. \n");

printf(" \n");

char c[1000];

printf("# Kuch to likh de yaar file me, copy karani hai file: \n");

scanf("%[^\n]s", c);

printf(" \n");

int size=0;

for(int i=0;i<100;i++)

{

if(c[i]=='\0'){

break;

}

else{

size++;

}

}

write(fd1, c, size);

printf("@ Given input text from user has been Successfully copied in file named 'My\_File.txt'. \n");

printf(" \n");

DIR \*dp = opendir("Your\_Directory");

int no\_of\_files=-2;

while(NULL != (dptr=readdir(dp)))

{

no\_of\_files++;

}

printf("# No. of files in directory which is named 'Your\_Directory' are: \n");

printf("%d\n",no\_of\_files);

printf(" \n");

int fd2 = open("Your\_Directory/My\_File\_Copy.txt", O\_CREAT | O\_RDWR, 0777);

printf("\* File named 'My\_File\_Copy.txt' has been created Successfully in Directory named 'Your\_Directory'. \n");

write(fd2, c, size);

printf("\* Content of File named 'My\_File' in Directory named 'My\_Directory' has been Successfully Copied to file named 'My\_File\_Copy.txt' which is in Directory named 'Your\_Directory' \n");

printf(" \n");

// -----------------------------------------------------------------------------------------

DIR \*dp1 = opendir("Your\_Directory");

int new\_no\_of\_files=-2;

while(NULL != (dptr=readdir(dp1)))

{

new\_no\_of\_files++;

}

printf("# Now no. of files in directory which is named 'Your\_Directory' are: \n");

printf("%d\n", new\_no\_of\_files);

printf(" \n");

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-5 (Process Management Using System Calls)

Q1. Write a program using system calls for operation on process to stimulate n fork calls to create (2^n - 1) child processes

---> Code:

#include<stdio.h> // for printf and scanf

#include<unistd.h> // for fork() & getpid()

#include<sys/types.h> // for fork() & getpid()

int main(){

int n;

printf("# Enter the no. of times you want to run the fork system call: ");

scanf("%d", &n);

for(int i=0; i<n; i++){

pid\_t r;

r = fork();

if(r==0){

printf("Current child process pid is %d \n", getpid());

}

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q2. Write a program using system calls for operations on processes to create a heirarchy of processes P1 -> P2 -> P3, also print the id and parent id for each process

---> Code:

#include<stdio.h> // for printf and scanf

#include<unistd.h> // for fork(), getpid() & getppid()

#include<sys/types.h> // for fork(), getpid() & getppid()

#include <stdlib.h> // for exit()

int main()

{

printf("Parent PID : %d \n", (int) getpid());

pid\_t pid = fork();

if(pid == 0)

{

printf("Child 1 PID : %d Parent PID : %d\n", (int) getpid(), (int) getppid());

pid\_t pid\_1 = fork();

if(pid\_1 == 0)

{

printf("Child 2 PID : %d Parent PID (Child 1) : %d \n", (int) getpid(), (int) getppid());

exit(0);

}

else

{

exit(0);

}

}

else

{

exit(0);

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q3. Write a program using system calls for operations on processes to create a heirarchy of processes: P3 <- P2 <- P1 -> P4 -> P5, also stimulate process P4 as orphan and P5 as zombie

---> Code:

#include<stdio.h> // for printf and scanf

#include<unistd.h> // for fork(), getpid(), sleep() & getppid()

#include<sys/types.h> // for fork(), getpid() & getppid()

#include<stdlib.h> // for exit()

// In Orphan Process, child process P4 goes to sleep and whenever the sleep time period is completed and P4 comes back for execution it's parent has already completed it's execution. So, it will get a garbage parent PID, whenever it wants to access it's parents PID as it is a orphan process.

// In Zombie Process, the parent process P4 goes to sleep and the child process P5 executes before P4 and leaves, and then after the completion of sleep time period process P4 executes, so here child process P5 will be called zombie process beacuse it has executed before it's parent's execution, and for P4 (parent) the process P5 is still visible in it's table but it has already completed it's execution.

int main()

{

printf("P1 PID : %d \n", (int) getpid());

pid\_t pid = fork();

if(pid == 0)

{

printf("P4 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());

printf("Child process P4 is sleeping \n");

pid\_t pid\_1 = fork();

sleep(5);

if(pid\_1 == 0)

{

printf("P5 PID : %d P4 PID : %d \n", (int) getpid(), (int) getppid());

printf("Zombie process P5's PID : %d \n", (int) getpid());

}

else{

printf("Orphan child process P4's PID : %d \n", (int) getpid());

printf("P4's New Parent PID : %d \n", (int) getppid());

}

}

else

{

pid = fork();

if(pid == 0)

{

printf("P2 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());

pid\_t pid\_1 = fork();

if(pid\_1 == 0)

{

printf("P3 PID : %d P2 PID : %d \n", (int) getpid(), (int) getppid());

exit(0);

}

else

{

exit(0);

}

}

else

{

exit(0);

}

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q4. Write a program using system calls for operations on processes to create a heirarchy of processes: P4 <- P3 <- P2 <- P1 -> P5 -> P6 -> P7, also stimulate process P4 as an orphan process and P7 as zombie process

---> Code:

#include<stdio.h> // for printf and scanf

#include<unistd.h> // for fork(), getpid(), sleep() & getppid()

#include<sys/types.h> // for fork(), getpid() & getppid()

#include<stdlib.h> // for exit()

int main()

{

printf("P1 PID : %d \n", (int) getpid());

pid\_t pid = fork();

if(pid == 0)

{

printf("P5 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getppid());

pid\_t pid\_1 = fork();

if(pid\_1 == 0)

{

printf("P6 PID : %d Parent P5 PID : %d \n", (int) getpid(), (int) getppid());

pid\_t pid\_2 = fork();

sleep(5);

if(pid\_2 == 0)

{

printf("Zombie process P7's PID: %d \n", (int) getpid());

printf("Parent P6 PID : %d \n", (int) getppid());

}

else

{

exit(0);

}

}

else

{

exit(0);

}

}

else

{

pid = fork();

if(pid == 0)

{

printf("P2 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getppid());

pid\_t pid\_1 = fork();

if(pid\_1 == 0)

{

printf("P3 PID : %d Parent P2 PID : %d \n", (int) getpid(), (int) getppid());

pid\_t pid\_2 = fork();

if(pid\_2 == 0)

{

sleep(3);

}

else

{

printf("Orphan child process P4's PID : %d \n", (int) pid\_2);

printf("P4's New Parent PID : %d \n", (int) getppid());

}

}

else

{

exit(0);

}

}

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-6 (Creation of Multithreaded Processes using Pthread Library)

Q1. Write a program using pthread to concatenate the strings, where multiple strings are passed to thread function

---> Code:

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<string.h>

char str1[100], str2[100];

char result[1000];

void \*concatenatestrings(){

strcat(result, str1);

strcat(result, str2);

pthread\_exit(NULL);

}

int main(){

pthread\_t thread;

printf("\* Enter the first string: ");

scanf("%s", str1);

printf("\* Enter the second string: ");

scanf("%s", str2);

pthread\_create(&thread, NULL, concatenatestrings, NULL);

pthread\_join(thread, NULL);

printf("@ Final result is: %s \n", result);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q2. Write a program using pthread to find the length of string, where strings are passed to thread function

---> Code:

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<string.h>

char length1[100];

int length=0;

void \*lengthstr(){

length=strlen(length1);

pthread\_exit(NULL);

}

int main(){

pthread\_t thread;

printf("\* Enter the String: ");

scanf("%[^\n]s", length1);

pthread\_create(&thread, NULL, lengthstr, NULL);

pthread\_join(thread, NULL);

printf("\* Total length of string is: %d \n", length);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q3. Write a program that performs statistical operations of calculating the average, maximum & minimum for a set of numbers. Create three threads where each performs their respective operations.

---> Code:

#include<stdio.h>

#include<pthread.h>

int arr[10] = {99, 22, 00, 88, 11, 102, 33, 66, 44, 55};

void \*sort(){

for(int i=0; i<10; i++){

for(int j=0; j<10; j++){

if(arr[i] < arr[j]){

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void \*min(){

int min = arr[0];

printf("\* Minimum element is = %d\n", min);

pthread\_exit(NULL);

}

void \*max(){

int max = arr[9];

printf("\* Maximum element is = %d \n", max);

pthread\_exit(NULL);

}

void \*avg(){

int sum=0;

for(int i=0;i<10;i++)

{

sum = sum + arr[i];

}

sum = sum/10;

printf("\* The average of the elements = %d \n", sum);

printf("\n");

pthread\_exit(NULL);

}

int main(){

printf("\n");

/\*

printf("Enter 10 elements in the array: ");

for(int i=0; i<10; i++)

{

scanf("%d", &arr[i]);

}

printf("\n");

\*/

printf("# Initial input array is: ");

for(int i=0; i<10; i++){

printf("%d ", arr[i]);

}

printf("\n");

pthread\_t sort\_thread, max\_thread, min\_thread, avg\_thread;

pthread\_create(&sort\_thread, NULL, sort, NULL);

pthread\_join(sort\_thread, NULL);

pthread\_create(&max\_thread, NULL, max, NULL);

pthread\_join(max\_thread, NULL);

pthread\_create(&min\_thread, NULL, min, NULL);

pthread\_join(min\_thread, NULL);

pthread\_create(&avg\_thread, NULL, avg, NULL);

pthread\_join(avg\_thread, NULL);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Q4. Write a multithreaded program where an array of integers is passed globally and is divided into two smaller lits and given as input to two threads. The thread will sort their half of the list and will pass the sorted list to a third thread which merges and sorts the list. The final sorted list is printed by the parent thread.

---> Code:

#include<stdio.h>

#include<pthread.h>

int arr[10] = {99, 22, 00, 88, 11, 100, 33, 66, 44, 55};

int arr\_first\_half[5], arr\_second\_half[5], final\_arr[10];

void \*final\_merge\_sort(){

for(int i=0; i<5; i++){

final\_arr[i] = arr\_first\_half[i];

final\_arr[i+5] = arr\_second\_half[i];

}

printf("# Merged array is: ");

for(int i=0; i<10; i++){

printf("%d ", final\_arr[i]);

}

printf("\n");

for(int i=0; i<10; i++){

for(int j=0; j<10; j++){

if(final\_arr[i] < final\_arr[j]){

int temp = final\_arr[i];

final\_arr[i] = final\_arr[j];

final\_arr[j] = temp;

}

}

}

printf("@ Final Merged & Sorted array is: ");

for(int i=0; i<10; i++){

printf("%d ", final\_arr[i]);

}

printf("\n");

printf("\n");

pthread\_exit(NULL);

}

void \*individual\_sort(){

for(int i=0; i<5; i++){

for(int j=0; j<5; j++){

if(arr\_first\_half[i] < arr\_first\_half[j]){

int temp = arr\_first\_half[i];

arr\_first\_half[i] = arr\_first\_half[j];

arr\_first\_half[j] = temp;

}

if(arr\_second\_half[i] < arr\_second\_half[j]){

int temp = arr\_second\_half[i];

arr\_second\_half[i] = arr\_second\_half[j];

arr\_second\_half[j] = temp;

}

}

}

pthread\_exit(NULL);

}

int main()

{

printf("\n");

/\*

printf("Enter 10 elements in the array: ");

for(int i=0; i<10; i++)

{

scanf("%d", &arr[i]);

}

printf("\n");

\*/

printf("# Initial input array is: ");

for(int i=0; i<10; i++){

printf("%d ", arr[i]);

}

printf("\n");

for(int i=0; i<5; i++){

arr\_first\_half[i] = arr[i];

arr\_second\_half[i] = arr[i+5];

}

pthread\_t parent\_thread;

pthread\_create(&parent\_thread, NULL, individual\_sort, NULL);

pthread\_join(parent\_thread, NULL);

printf("\* First half sorted array is: ");

for(int i=0; i<5; i++){

printf("%d ", arr\_first\_half[i]);

}

printf("\n");

printf("\* Second half sorted array is: ");

for(int i=0; i<5; i++){

printf("%d ", arr\_second\_half[i]);

}

printf("\n");

pthread\_create(&parent\_thread, NULL, final\_merge\_sort, NULL);

pthread\_join(parent\_thread, NULL);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-7th (Process Synchronization Using Semaphores/Mutex):

\* Race Around Condition:

---> Code:

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

int NUM\_THREADS = 2;

int shared = 0;

void \*thread\_func(void \*arg) {

int id = \*(int \*) arg;

int local = 0;

for (int i = 0; i < 1000000; i++) {

local = shared;

local++;

shared = local;

}

printf("Thread %d: shared = %d\n", id, shared);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, thread\_func, (void \*) &thread\_ids[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

printf("Final value of shared = %d\n", shared);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\* Race Around Condition Solved Using Mutex:

---> Code:

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

int NUM\_THREADS = 2;

int shared = 0;

pthread\_mutex\_t mutex;

void \*thread\_func(void \*arg) {

int id = \*(int \*) arg;

int local = 0;

pthread\_mutex\_lock(&mutex);

for (int i = 0; i < 1000000; i++) {

local = shared;

local++;

shared = local;

}

printf("Thread %d: shared = %d\n", id, shared);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

pthread\_mutex\_init(&mutex,NULL);

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, thread\_func, (void \*) &thread\_ids[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

printf("Final value of shared = %d\n", shared);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\* Race Around Condition Solved Using Semaphore:

---> Code:

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

int NUM\_THREADS = 2;

int shared = 0;

sem\_t semaphore;

void \*thread\_func(void \*arg) {

int id = \*(int \*) arg;

int local = 0;

sem\_wait(&semaphore);

for (int i = 0; i < 1000000; i++) {

local = shared;

local++;

shared = local;

}

printf("Thread %d: shared = %d\n", id, shared);

sem\_post(&semaphore);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

sem\_init(&semaphore,0,1);

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, thread\_func, (void \*) &thread\_ids[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

printf("Final value of shared = %d\n", shared);

sem\_destroy(&semaphore);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\* Producer & Consumer Using Mutex

---> Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

int BUFFER\_SIZE = 5;

int buffer[5];

int count = 0;

int last\_consumed\_index = 0;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t cond\_producer = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t cond\_consumer = PTHREAD\_COND\_INITIALIZER;

void\* producer(void\* arg) {

int item;

int iterations = 0;

while (iterations < 10) { // exit after 10 iterations

item = rand() % 100; // generate a random item

pthread\_mutex\_lock(&mutex);

if (count == BUFFER\_SIZE) {

pthread\_cond\_wait(&cond\_producer, &mutex);

}

if (count == 0) {

last\_consumed\_index = 0; // reset last consumed index if buffer is empty

}

buffer[last\_consumed\_index++] = item;

printf("Produced item: %d\n", item);

count++;

if (count == 1) {

pthread\_cond\_signal(&cond\_consumer);

}

pthread\_mutex\_unlock(&mutex);

iterations++;

}

return NULL;

}

void\* consumer(void\* arg) {

int item;

int iterations = 0;

while (iterations < 10) { // exit after 10 iterations

pthread\_mutex\_lock(&mutex);

if (count == 0) {

pthread\_cond\_wait(&cond\_consumer, &mutex);

}

item = buffer[--last\_consumed\_index];

printf("Consumed item: %d\n", item);

count--;

if (count == BUFFER\_SIZE - 1) {

pthread\_cond\_signal(&cond\_producer);

}

pthread\_mutex\_unlock(&mutex);

iterations++;

}

return NULL;

}

int main() {

pthread\_t producer\_thread, consumer\_thread;

srand(time(NULL)); // initialize the random seed

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\* Reader Writer Problem using Semaphore:

---> Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

int NUM\_READERS = 3;

int NUM\_WRITERS = 2;

int MAX\_ATTEMPTS = 5;

// Shared data

int shared\_data = 0;

int num\_readers = 0;

// Semaphores

sem\_t mutex;

sem\_t wrt;

// Reader function

void \*reader(void \*arg) {

int id = \*(int\*)arg;

int attempts = 0;

while (attempts < MAX\_ATTEMPTS) {

// Entry section

sem\_wait(&mutex);

num\_readers++;

if (num\_readers == 1) {

sem\_wait(&wrt);

}

sem\_post(&mutex);

// Critical section

printf("Reader %d read shared\_data as %d\n", id, shared\_data);

// Exit section

sem\_wait(&mutex);

num\_readers--;

if (num\_readers == 0) {

sem\_post(&wrt);

}

sem\_post(&mutex);

attempts++;

}

pthread\_exit(NULL);

}

// Writer function

void \*writer(void \*arg) {

int id = \*(int\*)arg;

int attempts = 0;

while (attempts < MAX\_ATTEMPTS) {

// Entry section

sem\_wait(&wrt);

// Critical section

shared\_data++;

printf("Writer %d wrote shared\_data as %d\n", id, shared\_data);

// Exit section

sem\_post(&wrt);

attempts++;

}

pthread\_exit(NULL);

}

int main() {

// Initialize semaphores

sem\_init(&mutex, 0, 1);

sem\_init(&wrt, 0, 1);

// Create reader threads

pthread\_t reader\_threads[NUM\_READERS];

int reader\_ids[NUM\_READERS];

for (int i = 0; i < NUM\_READERS; i++) {

reader\_ids[i] = i;

pthread\_create(&reader\_threads[i], NULL, reader, &reader\_ids[i]);

}

// Create writer threads

pthread\_t writer\_threads[NUM\_WRITERS];

int writer\_ids[NUM\_WRITERS];

for (int i = 0; i < NUM\_WRITERS; i++) {

writer\_ids[i] = i;

pthread\_create(&writer\_threads[i], NULL, writer, &writer\_ids[i]);

}

// Wait for threads to finish

for (int i = 0; i < NUM\_READERS; i++) {

pthread\_join(reader\_threads[i], NULL);

}

for (int i = 0; i < NUM\_WRITERS; i++) {

pthread\_join(writer\_threads[i], NULL);

}

// Destroy semaphores

sem\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

# Practical-8th (Inter Process Communication Using Pipes/Shared Memory/RCP)

\* Establish Interprocess communication (IPC) between Parent and child process using unnamed pipe.

---> Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

int pipefd[2]; // file descriptors for the pipe

char buffer[25];

pid\_t pid;

if (pipe(pipefd) == -1) { // create the pipe

printf("Pipe failed\n");

return 1;

}

pid = fork(); // create a child process

if (pid < 0) { // fork failed

printf("Fork failed\n");

return 1;

}

if (pid > 0) { // parent process

close(pipefd[0]); // close the read end of the pipe

printf("Parent process writing to pipe...\n");

write(pipefd[1], "Hello, child process!", 22);

close(pipefd[1]); // close the write end of the pipe

}

else { // child process

close(pipefd[1]); // close the write end of the pipe

printf("Child process reading from pipe...\n");

read(pipefd[0], buffer, 25);

printf("Child process received: %s\n", buffer);

close(pipefd[0]); // close the read end of the pipe

}

return 0;

}

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\* Establish Interprocess communication (IPC) between Parent and child process using named pipe.

---> Code:

#include <sys/types.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

int main(){

int pid, fd1, fd2;

char buffer[20];

mkfifo("my\_Pipe", 0666);

pid=fork();

if(pid > 0){

//Parent Section

fd1=open("my\_Pipe", O\_WRONLY);

write(fd1, "Hello Child Process\n",20);

}

if(pid==0){

//Child section

fd2=open("my\_Pipe", O\_RDONLY);

read(fd2, buffer, 20);

printf("%s", buffer);

}

return 0;

}

Exercise 1: Develop a program that demonstrates Inter-Process Communication (IPC) using named pipes. Tasks: Create a pair of named pipes: one for sending data and another for receiving data Develop a sender program that writes a message to the sending pipe. Create a receiver program that reads from the receiving pipe and displays the received message.=============================================================================================================================

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <string.h>

#define BUF\_SIZE 1024

int main() {

int sfd, rfd;

char \*s\_fifo = "/tmp/sender", \*r\_fifo = "/tmp/receiver", buf[BUF\_SIZE];

mkfifo(s\_fifo, 0666); mkfifo(r\_fifo, 0666);

sfd = open(s\_fifo, O\_WRONLY); rfd = open(r\_fifo, O\_RDONLY);

printf("Sender program\nEnter message to send: ");

fgets(buf, BUF\_SIZE, stdin); write(sfd, buf, strlen(buf) + 1);

printf("Receiver program\n");

read(rfd, buf, BUF\_SIZE); printf("Received: %s", buf);

close(sfd); close(rfd); unlink(s\_fifo); unlink(r\_fifo);

return 0;

}

Exercise 2: Demonstrate the usage of Shared Memory for IPC. Tasks:• Create a shared memory segment and attach it to multiple processes. • Develop a producer-consumer model, where one process writes data into the shared memory, and another process reads from it.=============================================================================================================================

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

#define SHM\_SIZE 1024

struct shared\_data { int written; char message[SHM\_SIZE]; };

int main() {

int shmid;

key\_t key = 1234;

struct shared\_data \*shared\_memory;

shmid = shmget(key, sizeof(struct shared\_data), IPC\_CREAT | 0666);

if (shmid == -1) { perror("shmget"); exit(EXIT\_FAILURE); }

shared\_memory = shmat(shmid, NULL, 0);

if (shared\_memory == (void \*)-1) { perror("shmat"); exit(EXIT\_FAILURE); }

printf("Producer process\nEnter message to write into shared memory: ");

fgets(shared\_memory->message, SHM\_SIZE, stdin);

shared\_memory->written = 1;

shmdt(shared\_memory);

printf("\nConsumer process\n");

while (!shared\_memory->written) { sleep(1); }

printf("Received message from shared memory: %s", shared\_memory->message);

shmctl(shmid, IPC\_RMID, NULL);

return 0;

}

Exercise 3: Explore IPC using Message Passing techniques. Tasks: Design two processes where one process sends a signal to another process. Develop signal havdlers in both processes to manage incoming signals and per- forin specific actions based on the received signal.=============================================================================================================================

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <signal.h>

#define MSG\_SIZE 128

struct message { long mtype; char mtext[MSG\_SIZE]; };

void sender\_handler(int sig) { printf("Sender received signal %d\n", sig); }

void receiver\_handler(int sig) { printf("Receiver received signal %d\n", sig); }

int main() {

pid\_t pid;

key\_t key = ftok("/tmp", 'A');

int msqid;

struct message msg;

msqid = msgget(key, IPC\_CREAT | 0666);

if (msqid == -1) { perror("msgget"); exit(EXIT\_FAILURE); }

pid = fork();

if (pid < 0) { perror("fork"); exit(EXIT\_FAILURE); }

else if (pid == 0) { signal(SIGUSR1, SIG\_IGN);

signal(SIGUSR2, sender\_handler); sleep(1);

msg.mtype = 1; sprintf(msg.mtext, "Hello from sender!");

if (msgsnd(msqid, &msg, sizeof(msg.mtext), 0) == -1) { perror("msgsnd");

exit(EXIT\_FAILURE); } }

else {

signal(SIGUSR1, receiver\_handler);

signal(SIGUSR2, SIG\_IGN);

printf("Receiver waiting for message...\n");

if (msgrcv(msqid, &msg, sizeof(msg.mtext), 1, 0) == -1) {

perror("msgrcv"); exit(EXIT\_FAILURE); }

printf("Received message from sender: %s\n", msg.mtext);

if (msgctl(msqid, IPC\_RMID, NULL) == -1) { perror("msgctl");

exit(EXIT\_FAILURE); } }

return 0;

}

################################################################### PEACE TE OUT ###################################################################